ELSEVIER

Computational Geometry 22 (2002) 119–142

Computational Geometry
Theory and Applications

www.elsevier.com/locate/comgeo

# Algebraic methods and arithmetic filtering for exact predicates on circle arcs ☆

Olivier Devillers *, Alexandra Fronville, Bernard Mourrain, Monique Teillaud

*INRIA Sophia Antipolis, 2004 Route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex, France*

Communicated by S. Fortune; received 1 July 2000; accepted 15 January 2001

**Abstract**

The purpose of this paper is to present a new method to design exact geometric predicates in algorithms dealing with curved objects such as circular arcs. We focus on the comparison of the abscissae of two intersection points of circle arcs, which is known to be a difficult predicate involved in the computation of arrangements of circle arcs. We present an algorithm for deciding the $x$-order of intersections from the signs of the coefficients of a polynomial, obtained by a general approach based on resultants. This method allows the use of efficient arithmetic and filtering techniques leading to fast implementation as shown by the experimental results. 

*Keywords:* Computational geometry; Geometric computing; Geometric predicate; Algebraic geometry; Bezoutian; Resultant; Arithmetic; Numerical precision; Filters

## 1. Introduction

Implementing geometric algorithms is difficult because the decisions made by such algorithms are taken on the basis of simple geometric questions, called *predicates*, solved by the evaluation of *continuous* functions subject to rounding errors, though the algorithms are basically of combinatorial and *discrete* nature. For example, the sweep line paradigm is a combinatorial algorithm relying on predicates such as $x$-comparisons.

The use of floating point arithmetic to evaluate predicates often produces inconsistencies. For instance, plane sweep algorithms, which are basic tools in computational geometry, are known to be very sensitive to numerical errors: when computing arrangements of curves, a plane sweep algorithm needs to sort intersection points between curves by $x$ coordinates, and if, due to erroneous numerical computations, the $x$ comparison test is not transitive, the algorithm may crash.

☆ This research was partially supported by the ESPRIT IV LTR Project No. 28155 (GALIA).

* Corresponding author.
*E-mail address:* olivier.devillers@sophia.inria.fr (O. Devillers).

0925-7721/01/$ – see front matter 
PII: S0925-7721(01)00050-5

To cope with this problem, people may either work on the combinatorial part and design new algorithms that support inconsistencies, or implement the predicates in an exact way so that the combinatorial algorithm may rely on them safely. In this paper, we use the second approach and concentrate our attention on predicates.

A predicate takes a continuous input (points, coefficients) and produces a discrete result; in general the result has three possible values: two main values (e.g., *inside, outside*) correspond to geometric situations where the answer remains the same in a neighborhood of the input, and the third value (e.g., *on the boundary*) is the situation, called *degenerate*, where the answer switches from one to the other main value. The general methodology that we propose here can be sketched as follows:

1. Determine the geometric configurations for which the input of the predicate is in a degenerate situation (same abscissae, cocircular points, ...).
2. Apply resultant techniques to compute a polynomial of the input parameters that characterizes these configurations.
3. Exploit the geometric meaning of this resultant polynomial in order to optimize the computations.
4. Deduce from the signs of the roots of this resultant polynomial (as a polynomial in one of the parameters) the value of the predicate.
5. Use efficient arithmetic and filtering techniques to get an efficient implementation of the predicate.

In this approach, the value of the predicate depends on a combination of signs of polynomials. The robustness of a geometric algorithm is clearly connected to the algebraic degrees of the polynomials involved in the predicates it uses. The maximum of these degrees has been proposed as a measure for algorithms [14].

We study this degree in the case of algorithms computing the intersections of curve segments.

Such algorithms are based on some of the following predicates [4]:

(a) $x$-order of endpoints;
(b) endpoint above or below curve;
(c) curve intersection test;
(d) orientation of three endpoints;
(e) $x$-order of endpoint and intersection point;
(f) intersect in slab;
(g) order of intersections on curve;
(h) $x$-order of intersections.

Predicates (a), (b), (c), (e), (h), basic for classical plane sweep algorithms, are represented in Fig. 1.

Recent researches have been performed to propose algorithms that use a restricted subset of predicates of low degrees [2–5]. Typically, one of the goals is to avoid predicates (e) and (h) because their degree is known to be higher than the degree of (a). This is true in particular for the most extensively studied case: the case of line segments.

We focus in this paper on the basic predicates (a), (e) and (h) and show in Section 2 that, in the case of circle arcs, for an appropriate representation of the data, they are equivalent. We prove that it is possible to compare exactly and efficiently the abscissae of two intersection points between circle arcs (predicate (h)), without computing these intersections.

To this aim, we use a general method based on resultants that allows us to find an algebraic expression of this predicate, involving polynomials of degree at most 12 in the data. We use the multivariate Bezoutian, introduced in Section 4, to construct this resultant [1,9]. This computation is performed in

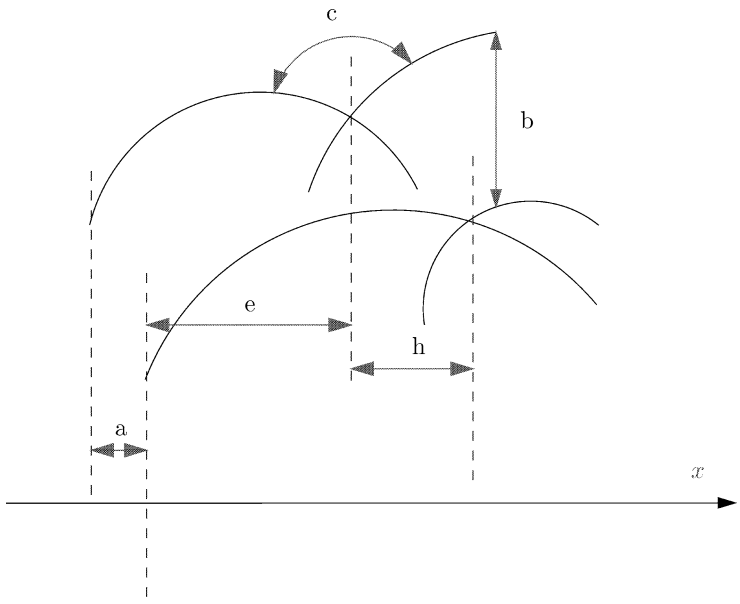



Fig. 1. Basic predicates for plane sweep.

Section 5 with the MAPLE package multires. [1] The resultant polynomial that we obtain corresponds to a geometric condition, which should be a function of basic intrinsic quantities (as asserted by the first fundamental theorem of invariants [21]). Indeed, moving back from algebra to geometry and using invariant theory [13,19], we express in Section 6 this resultant polynomial in terms of basic *invariants*, for which we give a geometric interpretation. This compact representation reduces the arithmetic complexity of the expressions whose sign must be evaluated and speeds up their numerical evaluation.

Then we develop in Section 7 a strategy to answer our predicate, from the signs of the coefficients of this resultant seen as a polynomial in one parameter, without necessarily evaluating completely the resultant. We show that, depending on the configuration of the data, computing the signs of polynomials of degrees 5, 6 or more, but never greater than 12, is sufficient.

The practical impacts of this approach are twofold. First, we show experimentally that our method, though more complicated to describe, substantially improves on the existing method (Section 3) of evaluation of the predicate, using the same kind of arithmetic. Second, this method allows us to use the most efficient filtering techniques [17], which does not apply with the other method. More precisely an implementation of an exact predicate usually requires two (or more) steps. In a first step, a *filter* (Section 8) computes approximations of the polynomials and if these values are far enough from 0, the signs of the polynomials can be determined safely; the higher the algebraic degree is, the more often the filter fails to conclude at this first step. In a second step, only in *close to* 0 cases, an exact computation is performed. We apply this technique here and report on the improvement that we obtain in Section 9. The different possibilities that we consider have been implemented in C++ with the CGAL library [2] (Computational Geometry Algorithms Library) [6].

[1] http://www.inria.fr/saga/logiciels/multires.html.

[2] http://www.cgal.org/.

## 2. Representation of circle arcs

Circles and also circle arcs can be defined in several ways. A circle can be described as passing through three points, or by its center and radius, or by its Cartesian equation.... For circle arcs the possibilities are even larger, for example an arc can be defined by its two endpoints and a third point on the arc in between.

However, in the computation of an arrangement of circle arcs, we need to construct new arcs having their endpoints defined as intersections of original arcs, and to represent these resulting arcs in the same way as the data in order to be able to use them for further computations. This implies some restrictions on the possible representations: using explicitly the endpoints in the arc representation would imply the possibility of computing exactly these endpoints. These exact computations are only possible using an arithmetic able of dealing with square roots, which would be very costly.

That is why we propose another representation for circle arcs, whose fundamental property is to be stable by the operation that consists in cutting arcs with other arcs: we define an arc to be supported by a circle and limited by two lines (with some additional orientation conditions to distinguish between all the arcs defined on a circle by two lines). The two endpoints are implicitly represented as intersections between the circle and two lines.

A circle $\mathcal{C}$ is defined by the coordinates of its center $\Omega = (\alpha, \beta)$ and its squared radius $\gamma$. Its equation is $\mathcal{C}(x, y) = 0$ where

$$\mathcal{C}(x, y) = (x - \alpha)^2 + (y - \beta)^2 - \gamma.$$

A line $\mathcal{L}$ is given by its equation $\mathcal{L}(x, y) = 0$ where

$$\mathcal{L}(x, y) = p\,x + q\,y + s.$$

(Parameters are supposed to be chosen so that circles, lines, and arcs are well defined.)

We assume the data to be represented exactly, for instance as fixed sized integers. Referring to the notion of algebraic degree, $\alpha$, $\beta$, $p$ and $q$ are of degree one and $\gamma$ and $s$ are of degree two.

In order to define unambiguously one vertex as $\mathcal{C} \cap \mathcal{L}$, we choose it to be the leftmost or the rightmost of the two intersection points (Fig. 2).

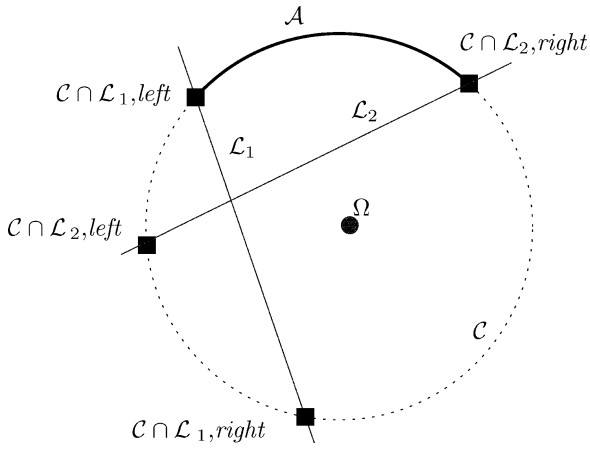



Fig. 2. Representation of vertices.

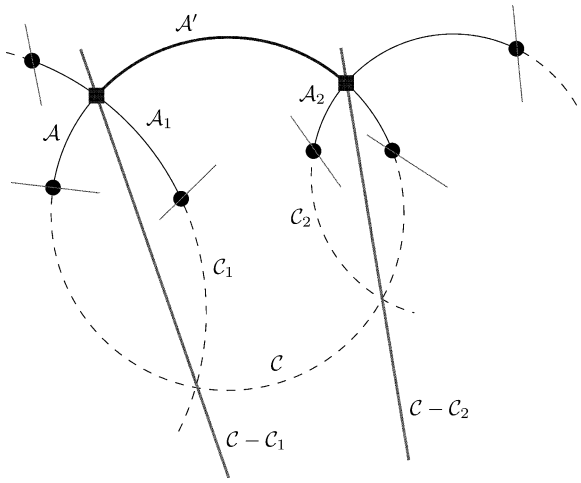



Fig. 3. Stability of the representation.

Let the arc $\mathcal{A}'$ be the result of cutting one data arc $\mathcal{A}$ supported by circle $\mathcal{C}$ by two other data arcs $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively supported by circles $\mathcal{C}_1$ and $\mathcal{C}_2$. Then $\mathcal{A}'$ has the following representation: $\mathcal{A}'$ is supported by the same circle $\mathcal{C}$, one of its two endpoints is defined as one of the intersections between $\mathcal{C}$ and the radical axis of $\mathcal{C}$ and $\mathcal{C}_1$, and its second endpoint is an intersection of $\mathcal{C}$ with the radical axis of $\mathcal{C}$ and $\mathcal{C}_2$ (Fig. 3).

Note that when two circles are given by their equations $\mathcal{C}(x, y) = (x-\alpha)^2 + (y-\beta)^2 - \gamma$ and $\mathcal{C}_1(x, y) = (x-\alpha_1)^2 + (y-\beta_1)^2 - \gamma_1$, then the equation of their radical axis is simply $\mathcal{C} - \mathcal{C}_1$, which has the important property that its coefficients are of degree one in the data. In this way, it can be represented exactly without increasing the degree of the coefficients.

Thus, predicates (a), (e), and (h) are equivalent with this representation.

## 3. Naive methods for $x$-comparison

Predicate (h) consists in comparing the abscissa of vertex $M_1$ (leftmost or rightmost) of an arc $\mathcal{A}_1$ with the abscissa of vertex $M_2$ of $\mathcal{A}_2$. $M_i$, $i = 1, 2$, is defined by $\mathcal{C}_i$ and $\mathcal{L}_i$ as in Section 2. Thus, the abscissa of $M_i$ is given by one (the smallest or the largest) of the solutions of the following second degree equation, obtained by eliminating $y$ from $\mathcal{C}_i$ and $\mathcal{L}_i$:

$$(p_i^2 + q_i^2)x^2 - 2(q_i^2\alpha_i - p_i q_i \beta_i - p_i s_i)x + (s_i^2 + 2q_i s_i \beta_i + q_i^2\alpha_i^2 + q_i^2\beta_i^2 - q_i^2\gamma_i) = 0. \tag{1}$$

To compare the two abscissae of $M_1$ and $M_2$, we can solve the two preceding equations, choose for each equation the correct solution, and compare them. The expressions of the solutions involve square roots.

This naive method will be compared in Section 9 with the new method that we propose in the sequel.

To avoid the computation of square roots, a natural idea [4] consists in squaring expressions. To evaluate the sign of an expression $E(\boldsymbol{u})$ with square roots, the idea is to write the equation $E(\boldsymbol{u}) = 0$. Then one square root is isolated on one side of the equality sign and both members of the equation are squared. The process is repeated as many times as necessary to eliminate all square roots and to obtain a polynomial $P(\boldsymbol{u}) = 0$. Unfortunately, this squaring process creates new roots, and the two formulations

are not equivalent: $E(\boldsymbol{u}) = 0 \Longrightarrow P(\boldsymbol{u}) = 0$ but $P(\boldsymbol{u}) = 0 \not\Longrightarrow E(\boldsymbol{u}) = 0$ and consequently the sign of $P(\boldsymbol{u})$ is not directly related to the sign of $E(\boldsymbol{u})$.

A correct and careful use of this squaring technique would require the introduction of extra polynomials to guarantee that expressions corresponding to square roots are actually non negative, which would make the naive method turn less naive and less easy to use and implement. In fact, it would give results similar to ours. Our method is as simple and more general. Moreover the resultant formulation systematizes this hand-made approach.

## 4. Resultants

In this section, we recall the basic results of resultant theory [3] [1,9].

The general situation of resultant theory is the case where we have a system of polynomial equations $\boldsymbol{f_c}(\boldsymbol{x}) = 0$ depending on parameters denoted by $\boldsymbol{c}$. Loosely speaking, the resultant is the necessary and sufficient condition (if it exists) on the parameters $\boldsymbol{c}$ such that the system of equations $\boldsymbol{f_c}(\boldsymbol{x}) = 0$ has a root $\boldsymbol{x}$ in a variety $X$. We will detail the case where $X$ is the projective space $\mathbb{P}^n$ hereafter, but it should be noticed that in practice we may need to consider other cases for $X$ [1]. In order to compute this resultant, we will introduce Bezoutian matrices which yield a way to compute it on general varieties $X$.

Considering the $N$ coefficients $\boldsymbol{c}$ of our system also as variables and denoting by $n$ the dimension of $X$, the system $\boldsymbol{f_c}(\boldsymbol{x}) = 0$ can be seen as a set of polynomial equations in a space of dimension $N + n$. The resultant is the condition on the $N$ coefficients $\boldsymbol{c}$ such that there exists a point $(\boldsymbol{c}, \boldsymbol{x})$ satisfying this system of equations $\boldsymbol{f_c}(\boldsymbol{x}) = 0$. In other words, the resultant eliminates the variables $\boldsymbol{x}$. This is why it was originally called "polynôme éliminant" [16]. Geometrically speaking, it is the equation of the projection of the set of solutions from the space of dimension $N+n$ to the space of coefficients of dimension $N$. That is also the reason why it has many applications in Effective Algebraic Geometry [9]. One of them is of course polynomial system solving by projecting the set of solutions on a line and by solving a univariate polynomial (or equivalently an eigenvalue problem). In this paper, we present yet another application of resultants to computational geometry and more precisely to the design of certified geometric predicates.

### 4.1. Resultant over $\mathbb{P}^n$

The situation that we will need to consider is the classical case of resultant over the projective space. We consider the $n + 1$ homogeneous polynomials:

$$\boldsymbol{f_c}(\boldsymbol{x}) = \begin{cases} f_0(\boldsymbol{x}) = \sum_{|\alpha|=d_0} c_{0,\alpha} \boldsymbol{x}^{\alpha}, \\ \vdots \\ f_n(\boldsymbol{x}) = \sum_{|\alpha|=d_n} c_{n,\alpha} \boldsymbol{x}^{\alpha}, \end{cases} \tag{2}$$

where

- $\boldsymbol{c} = (c_{i,\alpha})_{i,\alpha}$ are parameters,
- $\boldsymbol{x} = (x_0, \ldots, x_n)$ is a point of the projective space $\mathbb{P}^n$ of dimension $n$,
- for $\alpha = (\alpha_0, \ldots, \alpha_n) \in \mathbb{N}^n$, $\boldsymbol{x}^{\alpha}$ denotes $x_0^{\alpha_0} \cdots x_n^{\alpha_n}$ and $|\alpha| = \alpha_0 + \cdots + \alpha_n$.

[3] The reader who is not familiar with this subject may skip this part, but he must be aware that sooner or later he may have to come across it again.

**Definition 1.** The projective *resultant* $Res_{\mathbb{P}^n}(\boldsymbol{f}_{\boldsymbol{c}})$ is the necessary and sufficient condition on $\boldsymbol{c}$ such that the homogeneous polynomials $f_0, \ldots, f_n$ have a common root in $\mathbb{P}^n$.

It is a multi-homogeneous polynomial of degree $\prod_{j\neq i} d_j$ in the coefficients of each $f_i$.

The computation of resultants typically relies on obtaining matrices whose determinant is either the exact resultant polynomial or, more generally, a nontrivial multiple of it. In addition, for solving polynomial systems these matrices are sufficient, since they reduce the given nonlinear problem to a question in linear algebra. In the next section, we present one construction of such matrices, which has the advantage to apply for a large class of resultants.

### *4.2. Multivariate Bezoutians*

The multivariate Bezoutian gives a method to compute the resultant over $\mathbb{P}^n$ as explained now. One of the advantages of this tool, compared with Macaulay or the Toric formulations is that it can be generalized easily to a wide range of varieties [1]. We recall here its definition, which generalizes the univariate case considered by Bézout [8].

**Definition 2.** Let $\tilde{f}_0, \tilde{f}_1, \ldots, \tilde{f}_n$ be $n+1$ polynomials in the variables $\tilde{\boldsymbol{x}} = (x_1, \ldots, x_n)$ with coefficients in a ring $\mathbb{K}$. Their *Bezoutian* $\Theta_{\tilde{f}_0,\tilde{f}_1,\ldots,\tilde{f}_n}$ is the polynomial in $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{y}}$ defined by:

$$\Theta_{\tilde{f}_0,\tilde{f}_1,\ldots,\tilde{f}_n}(\tilde{\boldsymbol{x}},\tilde{\boldsymbol{y}}) = \begin{vmatrix} \tilde{f}_0(\tilde{\boldsymbol{x}}) & \Theta_1(\tilde{f}_0)(\tilde{\boldsymbol{x}},\tilde{\boldsymbol{y}}) & \ldots & \Theta_n(\tilde{f}_0)(\tilde{\boldsymbol{x}},\tilde{\boldsymbol{y}}) \\ \vdots & \vdots & & \vdots \\ \tilde{f}_n(\tilde{\boldsymbol{x}}) & \Theta_1(\tilde{f}_n)(\tilde{\boldsymbol{x}},\tilde{\boldsymbol{y}}) & \ldots & \Theta_n(\tilde{f}_n)(\tilde{\boldsymbol{x}},\tilde{\boldsymbol{y}}) \end{vmatrix},$$

where

$$\Theta_i(\tilde{f}_j)(\tilde{\boldsymbol{x}},\tilde{\boldsymbol{y}}) = \frac{\tilde{f}_j(y_1,\ldots,y_{i-1},x_i,\ldots,x_n) - \tilde{f}_j(y_1,\ldots,y_i,x_{i+1},\ldots,x_n)}{x_i - y_i}$$

for $i = 1, \ldots, n$ and $j = 0, \ldots, n$.

Let $\Theta_{\tilde{f}_0,\tilde{f}_1,\ldots,\tilde{f}_n}(\tilde{\boldsymbol{x}},\tilde{\boldsymbol{y}}) = \sum \lambda_{\alpha,\beta}\tilde{\boldsymbol{x}}^{\alpha}\tilde{\boldsymbol{y}}^{\beta}, \lambda_{\alpha,\beta} \in \mathbb{K}$, be the decomposition of the Bezoutian. We order the monomials that appear in $\Theta_{\tilde{f}_0,\tilde{f}_1,\ldots,\tilde{f}_n}$. The *Bezoutian matrix* of $\tilde{f}_0, \ldots, \tilde{f}_n$ is the matrix $B_{\tilde{f}_0,\ldots,\tilde{f}_n} = (\lambda_{\alpha,\beta})_{\alpha,\beta}$. Its entries are in $\mathbb{K}$.

The Bezoutian was used by Bézout to construct the resultant of two polynomials in one variable. It is possible to recover the general resultant of $n+1$ homogeneous polynomials $f_0, \ldots, f_n$ over $\mathbb{P}^n$, from the Bezoutian matrices, as shown by the next theorem.

**Theorem 3.** *For $i = 0, \ldots, n$, let $\tilde{f}_i(x_1, \ldots, x_n) = f_i(1, x_1, \ldots, x_n)$. Any maximal non-zero minor of the Bezoutian matrix $B_{\tilde{f}_0,\ldots,\tilde{f}_n}$ is divisible by the resultant $Res_{\mathbb{P}^n}(f_0, \ldots, f_n)$.*

**Proof.** The proof is a particular case of Theorem 3.4 of Busé et al. [1]. (We use the polynomial map $\sigma : \mathbb{K}^n \to \mathbb{P}^n$ such that $(t_1, \ldots, t_n) \mapsto (1 : t_1 : \ldots : t_n)$.) □

In practice, in order to compute a maximal minor of $B_{\tilde{f}_0,\ldots,\tilde{f}_n}$, we apply the fraction-free Gaussian elimination [11] also known as Bareiss method. This method gives a matrix in a triangular form whose

coefficients in the last non-zero line are maximal minors of the original matrix. By computing the GCD of the non-zero elements of this last line, we get a multiple of the resultant.

We illustrate this construction by a small example, worked out in Maple with the package multires

```
> read multires;
> f0 := u[0]+u[1]*x[1]+u[2]*x[2];
> f1 := 13*x[1]^2+8*x[1]*x[2]+4*x[2]^2-8*x[1]-8*x[2]+2;
> f2 := x[1]^2+x[1]*x[2]-x[1]-1/6;
> Theta([f0,f1,f2],[x[1],x[2]]);
```

Here is the polynomial in $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{y}}$:

$$\begin{aligned}
&-4u_0y_1y_2+5u_0{y_1}^2+(4u_0-\tfrac{2}{3}u_1)y_2+(8u_0-\tfrac{10}{3}u_1+\tfrac{25}{6}u_2)y_1+(\tfrac{4}{3}u_1-8u_0-\tfrac{10}{3}u_2)\\
&\quad+(-4u_1y_1y_2+5u_1{y_1}^2-4u_0y_2+(8u_1+5u_2+5u_0)y_1+(8u_0+\tfrac{25}{6}u_2))x_1\\
&\quad+(-4u_2y_1y_2+5u_2{y_1}^2+(4u_2-4u_0)y_2+(8u_1-4u_0)y_1+(\tfrac{10}{3}u_2+12u_0-\tfrac{2}{3}u_1))x_2\\
&\quad+(-4u_2y_2-4u_1y_1-4u_0)x_1x_2+(-4u_2y_2-4u_1y_1-4u_0){x_2}^2
\end{aligned}$$

and the corresponding Bezoutian matrix:

```
> mbezout([u[0]+u[1]*x[1]+u[2]*x[2],f1,f2],[x[1],x[2]]);
```

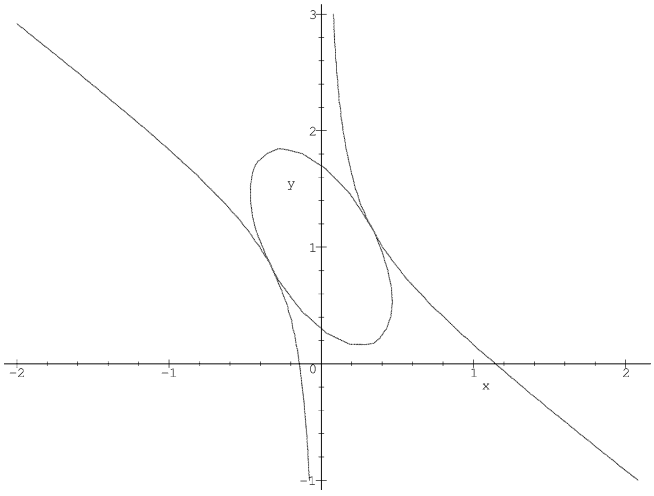
$$\begin{bmatrix}
-4u_0 & 5u_0 & 4u_0-\frac{2}{3}u_1 & 8u_0-\frac{10}{3}u_1+\frac{25}{6}u_2 & \frac{4}{3}u_1-8u_0-\frac{10}{3}u_2\\
-4u_1 & 5u_1 & -4u_0 & 8u_1+5u_2+5u_0 & 8u_0+\frac{25}{6}u_2\\
-4u_2 & 5u_2 & -4u_0+4u_2 & 8u_1-4u_0 & -\frac{2}{3}u_1+\frac{10}{3}u_2+12u_0\\
0 & 0 & -4u_2 & -4u_1 & -4u_0\\
0 & 0 & -4u_2 & -4u_1 & -4u_0
\end{bmatrix}.$$

One of its non-zero maximal minors is:

```
> factor(det(submatrix(",1..4,2..5)));
```

$$\tfrac{5}{11664}\left(u_0+\tfrac{1}{3}u_1+\tfrac{7}{6}u_2\right)^2\left(u_0-\tfrac{1}{3}u_1+\tfrac{5}{6}u_2\right)^2. \tag{3}$$

In this particular case, the vanishing of this polynomial is the necessary and sufficient condition on the parameters $\boldsymbol{u}$ such that $f_0$, $f_1$, $f_2$ have a common root. Indeed the geometric picture for the polynomials $f_1$, $f_2$ is as follows:




Their common points are $(\frac{1}{3}, \frac{7}{6})$, $(-\frac{1}{3}, \frac{5}{6})$ with multiplicity 2 and the polynomial (3) is just the condition that the linear form $f_0$ passes through one of these points, taking into account their multiplicities.

## 5. Application of resultants to $x$-comparison of intersections

We now illustrate the use of resultants for the arrangement of circle arcs. This method can be generalized to many other situations involving algebraic objects (for instance conics). The predicate that we are considering is the relative position of the abscissae of the vertices of two circle arcs. The primal idea is that the relative position of the vertices can be decided from the sign of a polynomial in the input parameters which vanishes when two of these points have the same abscissae. In fact, this idea does not yield directly the required polynomial and needs to be elaborated a little more as follows: we introduce a new parameter of translation $t$ along the $x$-axis as one of the input parameters. Then, we compute the resultant of the polynomial equations corresponding to equal abscissae using the Bezoutian formulation, and we obtain a polynomial $P(t)$ whose coefficients are polynomials in the parameters of the circle arcs.

As we will see, this polynomial is of degree 4 with 4 reals roots, which is not surprising since there are obviously 4 translations such that one of the abscissae of one arc coincides with one of the abscissae of the second arc. The signs of the roots of this polynomial will give us the relative configurations of the two arcs. We will use the sign of the coefficients of this polynomial in $t$ to determine the signs of the roots and thus the configuration of these two circle arcs.

This yields a method for $x$-comparison, in which the signs of some polynomial coefficients must be evaluated. This method, which seems to be more complicated than the naive one (Section 3), allows us to use static and semi-static filters and thus to speed up the computation as we will see. Moreover it can be generalized to higher degree curves.

As explained previously, we must compare the abscissae of two vertices of the circle arcs $\mathcal{A}_1$ and $\mathcal{A}_2$. Since a vertex is chosen on each arc, we can consider only one line per arc: the line defining the vertex we are interested in. The vertex is thus defined as the leftmost or rightmost intersection of circle $\mathcal{C}_i(x, y) = 0$ and line $\mathcal{L}_i(x, y) = 0$. More precisely, $M_i^l$ (resp. $M_i^r$) denotes the leftmost (resp. rightmost) intersection between $\mathcal{C}_i$ and $\mathcal{L}_i$, for $i = 1, 2$, and $l_i$ (resp. $r_i$) denotes the abscissa of $M_i^l$ (resp. $M_i^r$) (see Fig. 4).

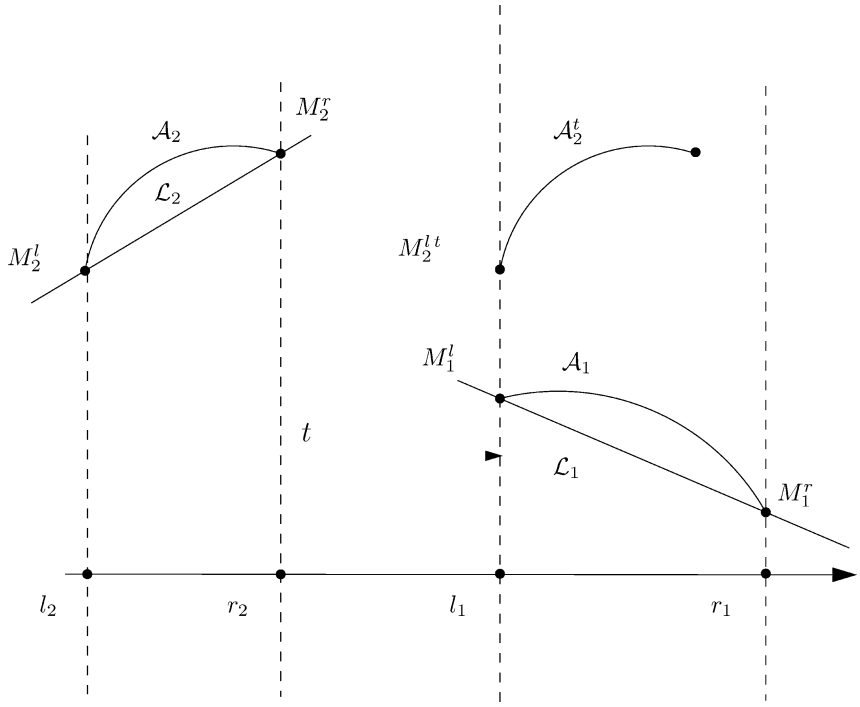



Fig. 4. Translation.

We translate the arc $\mathcal{A}_1$ by $t$ in the $x$ direction. The endpoints of this translated arc $\mathcal{A}_1^t$ are defined by the equations $\mathcal{C}_1(x+t, y) = 0$ and $\mathcal{L}_1(x+t, y) = 0$. We remark that two of the abscissae of the vertices of $\mathcal{A}_1^t$ and $\mathcal{A}_2$ coincide if and only if the system

$$\begin{cases} \mathcal{C}_1(x+t, y) = 0 \\ \mathcal{L}_1(x+t, y) = 0 \\ \mathcal{C}_2(x, z) = 0 \\ \mathcal{L}_2(x, z) = 0 \end{cases}$$

has a solution in $x, y, z$. It is a system of 4 equations in 3 unknowns, for which the resultant theory over $\mathbb{P}^n$ can be applied (see Section 4). To compute this resultant, we first compute the Bézout matrix with Maple using the multires package.

```
> C1 := (x+t-alpha1)^2 + (y-beta1)^2 - gamma1:
> L1 := p1*(x+t) + q1*y + s1:
> C2 := (x-beta2)^2 + (z-beta2)^2 - gamma2:
> L2 := p2*x + q2*z + s2:
> Bez := mbezout([C1,L1,C2,L2],[x,y,z]);
```

Bez is a $7 \times 7$ matrix in the parameters $(\alpha_1, \beta_1, \alpha_2, \beta_2, \gamma_1, \gamma_2, p_1, p_2, q_1, q_2, s_1, s_2)$, on which we apply fraction-free Gaussian elimination.

```
> Gau := ffgausselim(Bez);
```

The matrix `Gau` has the following form:

$$\begin{bmatrix} * & * & \cdots & * & * \\ 0 & * & \cdots & * & * \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & G_{n,n} & * \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The element $G_{n,n}$ is a maximal minor of the Bézout matrix and therefore a multiple of the resultant. Factoring out $G_{n,n}$ and removing the extraneous factors, we get an irreducible polynomial $P(t)$, which is of degree 4 in $t$. Consequently it is the resultant of the polynomial system.

$$P(t) = P_0t^4 + P_1t^3 + P_2t^2 + P_3t + P_4.$$

$P_0, \ldots, P_4$ are polynomials of respective degrees 8, 9, 10, 11 and 12, in the parameters $\alpha_1, \beta_1, \alpha_2, \beta_2, \gamma_1, \gamma_2, p_1, p_2, q_1, q_2, s_1, s_2$ where $\gamma_1, \gamma_2, s_1,$ and $s_2$ are considered of degree 2 (see Section 2). The expansion of $P$ contains 659 monomials. It is not given here.

**Remark.** Another approach would have consisted in computing the quadratic equation giving the abscissae of the vertex for each circle arc, as in Section 3. It is obtained by elimination of $y$ in the equations $\mathcal{C}_i(x, y) = 0$ and line $\mathcal{L}_i(x, y) = 0$, using the classical Sylvester resultant, which yields two equations of the form

$$Q_i(x, 1) = 0, \quad i = 1, 2$$

(see Eq. (1) in Section 3), where

$$Q_i(v, w) = A_iv^2 - 2B_ivw + C_iw^2, \quad i = 1, 2. \tag{4}$$

Eliminating $x$ in these two quadratic equations (again using Sylvester resultant) also yields a multiple of the polynomial $P$. Though this approach is conceptually simpler to understand, it does not apply for general geometric predicates. On the contrary the methodology that we describe here, which eliminates the variables in one step, can be generalized to other cases, which explains why we presented these general tools.

## 6. Reducing the degrees

In this section, we show that classical invariant theory considerations allow us to rewrite the polynomials $P_i$, $i = 0, \ldots, 4$, in a simpler way.

The resultant is known to be the polynomial in the input parameters having minimal degree among the polynomials giving conditions so that the endpoints abscissae are equal. However, its coefficients can be expressed in a more compact form. We will see in Section 7 that the signs of the coefficients play a central role in the method. These signs will be trivially deduced from the signs of the factors.

Using Maple, we can simplify the expressions of the coefficients $P_i$, $i = 0, \ldots, 4$:

$$
\begin{aligned}
P_0 &= A_1^2 A_2^2,\\
P_1 &= 4A_1 A_2 J,\\
P_2 &= 4J^2 + 2A_1 A_2 K,\\
P_3 &= 4JK,\\
P_4 &= K^2 - 4I_1 I_2 = -4JJ' + J''^2,
\end{aligned}
$$

where:

$$
\begin{aligned}
-1- \quad & A_1 = p_1^2 + q_1^2,\\
& A_2 = p_2^2 + q_2^2,\\
-2- \quad & B_1 = q_1^2\alpha_1 - p_1 s_1 - p_1 q_1 \beta_1,\\
& B_2 = q_2^2\alpha_2 - p_2 s_2 - p_2 q_2 \beta_2,\\
-3- \quad & C_1 = s_1^2 + 2q_1 s1\beta_1 + q_1^2\alpha_1^2 + q_1^2\beta_1^2 - q_1^2\gamma_1,\\
& C_2 = s_2^2 + 2q_2 s_2\beta_2 + q_2^2\alpha_2^2 + q_2^2\beta_2^2 - q_2^2\gamma_2
\end{aligned}
$$

are the coefficients of the two binary forms $Q_1(v,w)$ and $Q_2(v,w)$ defined by Eq. 4 in Section 5, and:

$$
\begin{aligned}
-4- \quad & I_1 = B_1^2 - A_1 C_1,\\
& I_2 = B_2^2 - A_2 C_2,\\
-5- \quad & J = A_1 B_2 - A_2 B,\\
& J' = B_1 C_2 - B_2 C_1,\\
& J'' = A_1 C_2 - A_2 C_1,\\
-6- \quad & K = C_1 A_2 + A_1 C_2 - 2B_1 B_2.
\end{aligned}
$$

The polynomials $I_1$, $I_2$ and $K$ are the classical invariants [7,15,19] by the action of $SL_2(\mathbb{C})$ (subgroup of $GL(\mathbb{C}^2)$ of matrices of determinant 1) of $Q_1(v,w)$ and $Q_2(v,w)$.

The polynomial $J$ (resp. $J'$) is an invariant of the same forms by translations $(v,w) \mapsto (v + aw, w)$ (resp. $(v,w) \mapsto (v, w + bv)$). Though $J''$ is not an invariant, this notation is used because of the similarity of its expression with those of $J$ and $J'$.

With this representation, we now need 80 arithmetic operations in order to evaluate $P(t)$ instead of $659 \times 13 = 9048$ arithmetic operations for the initial monomial expansion.

*Geometric interpretation of the algebraic expressions*

Though these expressions are obtained by algebraic methods, they still have a geometric meaning.

$-1-$ Interpreting $A_i$, $i = 1, 2$, is straightforward: it is the squared norm of the vector $(p_i, q_i)$ orthogonal to line $\mathcal{L}_i$.

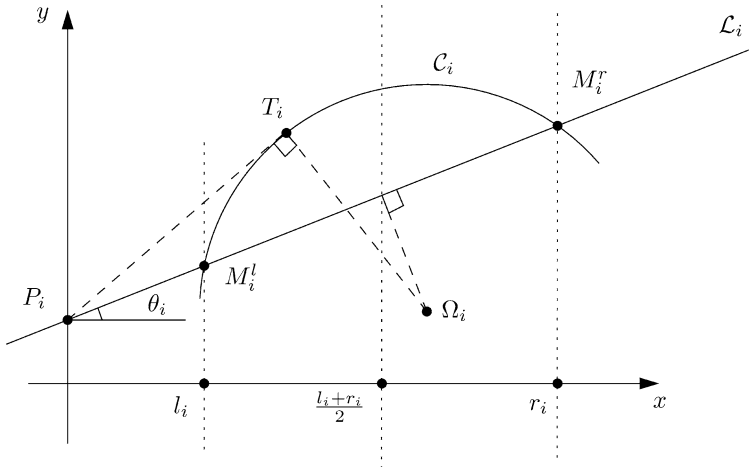



Fig. 5. Notation.

$-2,3-$ As already noticed, $Q_i(x,1) = A_i x^2 - 2B_i x + C_i$ is the polynomial whose roots are the abscissae $l_i$ and $r_i$. So, we get trivially:

$$\frac{B_i}{A_i} = \frac{l_i + r_i}{2} \quad \text{and} \quad \frac{C_i}{A_i} = l_i \cdot r_i.$$

Looking back to the data, $B_i/A_i$ can also be seen as the abscissa of the projection of the center $\Omega_i$ of $\mathcal{C}_i$ onto $\mathcal{L}_i$.

It is interesting to notice that $C_i/A_i$ is related to the power of the point $P_i$, intersection between $\mathcal{L}_i$ and the $y$ axis, with respect to circle $\mathcal{C}_i$. Indeed, with the notation defined in Fig. 5, we have

$$\begin{aligned}\frac{C_i}{A_i} &= l_i \cdot r_i \\ &= \|P_i M_i^r\| \cdot \|P_i M_i^l\| \cdot \cos^2\theta_i \\ &= \|P_i M_i^r\| \cdot \|P_i M_i^l\| \cdot \frac{q_i^2}{p_i^2 + q_i^2},\end{aligned}$$

thus

$$C_i = \text{power}(P_i, \mathcal{C}_i) \cdot q_i^2,$$

which can also be written as

$$C_i = \|P_i T_i\|^2 \cdot q_i^2$$

if $P_i$ is outside circle $\mathcal{C}_i$.

$-4-$ Let us now examine the invariants $I_i = B_i^2 - A_i C_i$, $i = 1, 2$. They depend respectively on one circle arc: $I_i$ is the discriminant of $Q_i$. Let $d(\Omega_i, \mathcal{L}_i)$ be the distance from the center $\Omega_i = (\alpha_i, \beta_i)$ of $\mathcal{C}_i$ to $\mathcal{L}_i$. We have

$$d(\Omega_i, \mathcal{L}_i)^2 = \frac{(p_i\alpha_i + q_i\beta_i + s_i)^2}{p_i^2 + q_i^2}.$$

The reader will easily check that

$$I_i = (\gamma_i - d(\Omega_i, \mathcal{L}_i)^2) A_i q_i^2$$

from which it is clear that $\mathcal{C}_i$ and $\mathcal{L}_i$ intersect if and only if $I_i > 0$ (remember that $\gamma_i$ is the squared radius of $\mathcal{C}_i$) which is necessary for the circle arc to be defined. We also notice that $I_i = 0$ if and only if $\mathcal{L}_i$ is tangent to $\mathcal{C}_i$, i.e., $M_i^l = M_i^r$ or $r_i = l_i$, which appears clearly in the following expression:

$$\frac{\sqrt{I_i}}{A_i} = \frac{r_i - l_i}{2}.$$

−5− The polynomial $J$ is an invariant relating the two forms $Q_1$ and $Q_2$.

$$\begin{aligned} J &= A_1 B_2 - A_2 B_1 \\ &= A_1 A_2 \left( \frac{B_2}{A_2} - \frac{B_1}{A_1} \right) \\ &= A_1 A_2 \left( \frac{l_2 + r_2}{2} - \frac{l_1 + r_1}{2} \right). \end{aligned}$$

Thus $J/(A_1 A_2)$ is the signed distance between the respective projections of the midpoints of $[M_1^l, M_1^r]$ and $[M_2^l, M_2^r]$ onto the horizontal axis.

The invariant $J'$ is obtained from $Q_1(v, w)$ and $Q_2(v, w)$ in the same way as $J$, exchanging the roles of $v$ and $w$. So,

$$\begin{aligned} J' &= C_1 C_2 \left( \frac{\frac{1}{l_1} + \frac{1}{r_1}}{2} - \frac{\frac{1}{l_2} + \frac{1}{r_2}}{2} \right) \\ &= A_1 A_2 l_1 r_1 l_2 r_2 \left( \frac{\frac{1}{l_1} + \frac{1}{r_1}}{2} - \frac{\frac{1}{l_2} + \frac{1}{r_2}}{2} \right). \end{aligned}$$

Though $J''$ is not an invariant, it can be easily expressed in terms of the abscissae of the arcs endpoints.

$$\begin{aligned} J'' &= A_1 C_2 - A_2 C_1 \\ &= A_1 A_2 \left( \frac{C_2}{A_2} - \frac{C_1}{A_1} \right) \\ &= A_1 A_2 (l_2 r_2 - l_1 r_1). \end{aligned}$$

−6− Finding a simple geometric meaning for $K$ is more tricky. As $J$, $K$ depends on both $Q_1$ and $Q_2$.

$$\begin{aligned} K &= C_1 A_2 + A_1 C_2 - 2 B_1 B_2 \\ &= A_1 A_2 \left( \frac{C_1}{A_1} + \frac{C_2}{A_2} - 2 \frac{B_1}{A_1} \frac{B_2}{A_2} \right) \\ &= A_1 A_2 \left( l_1 r_1 + l_2 r_2 - 2 \frac{l_1 + r_1}{2} \frac{l_2 + r_2}{2} \right). \end{aligned}$$

It can be noticed [7] that $K = 0$ if and only if the points $(l_1, r_1, l_2, r_2)$ on the $x$ axis form an harmonic division. Indeed, if we denote by $[l_1, r_1; l_2, r_2]$ the cross ratio of the four points in this order, the reader will easily check that

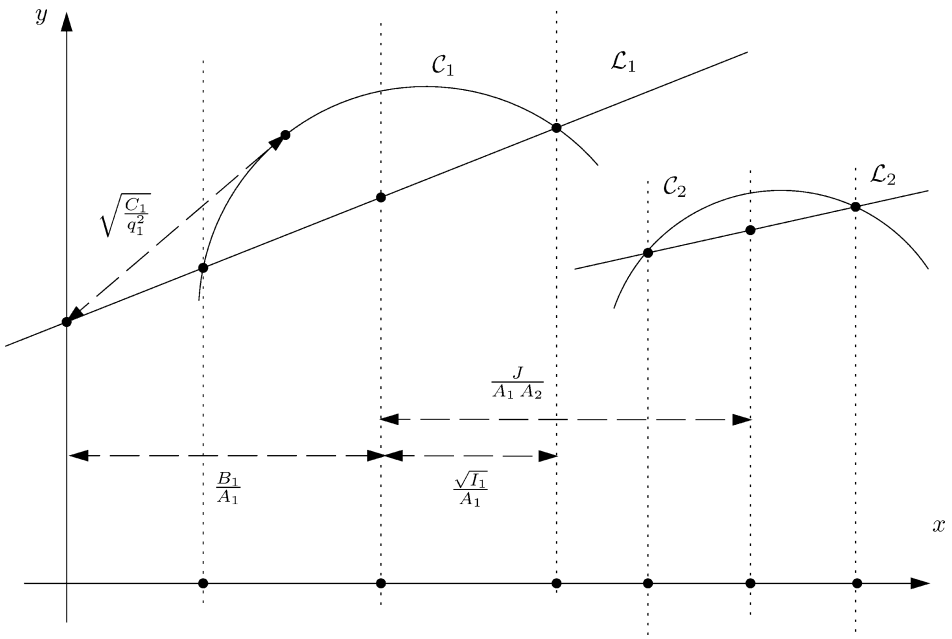



Fig. 6. Geometric interpretation.

$$[l_1, r_1; l_2, r_2] = \frac{(l_2 - l_1)(r_2 - r_1)}{(l_2 - r_1)(r_2 - l_1)} \quad \text{(definition)}$$
$$= \frac{K + 2\sqrt{I_1 I_2}}{K - 2\sqrt{I_1 I_2}}$$

which gives

$$K = 0 \Longleftrightarrow [l_1, r_1; l_2, r_2] = -1.$$

It is worth noticing that, since $P_4 = K^2 - 4I_1 I_2$, the sign of $P_4$ is the same as the sign of $[l_1, r_1; l_2, r_2]$. So, the sign of $P_4$ can be interpreted in terms of the ordering of the points $l_1, r_1, l_2, r_2$ on the $x$-axis.

## 7. Resultant and comparison of abscissae

We are now given a polynomial $P$ whose four roots give the translations that make the abscissae of an endpoint of $\mathcal{A}_1$ and an endpoint of $\mathcal{A}_2$ coincide.

We classify here the different possible configurations of the two arcs (see Fig. 7) and relate them with the number of positive roots of $P$.

*Case* 1 $l_1 < r_1 < l_2 < r_2$ 4 positive roots.

*Case* 2 $l_1 < l_2 < r_1 < r_2$ 3 positive roots.

*Case* 3a $l_1 < l_2 < r_2 < r_1$ 2 positive roots.

*Case* 3b $l_2 < l_1 < r_1 < r_2$ 2 positive roots.

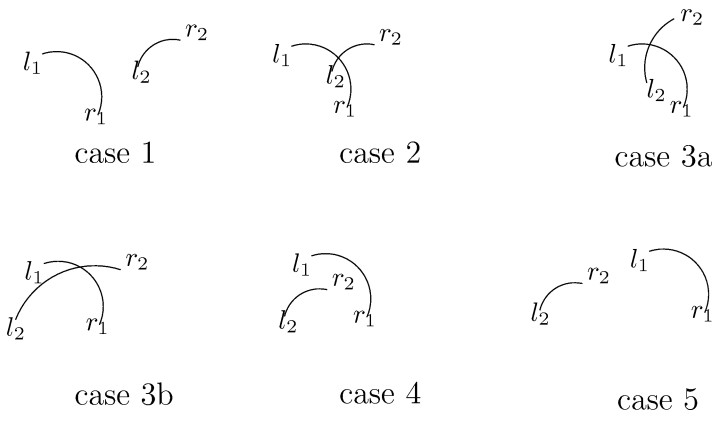



Fig. 7. All the configurations.

*Case* 4 $l_2 < l_1 < r_2 < r_1$ 1 positive root.

*Case* 5 $l_2 < r_2 < l_1 < r_1$ 0 positive root.

Thus, except for Cases 3a and 3b, the number of positive roots of $P$ gives all the necessary information on the $x$-order of the endpoints of the arcs.

We now remark that we are only interested in comparing two abscissae, say $l_1$ and $l_2$, so the complete determination of the case is not necessary, since $l_1 < l_2$ if we are in Cases 1, 2 or 3a and $l_2 < l_1$ otherwise.

Descartes rule specifies that the number of sign changes in the coefficients $P_0$, $P_1$, $P_2$, $P_3$ and $P_4$ of $P$ is an upper bound for the number of positive roots of $P$ [20]. In the case where all the roots of $P$ are real, Descartes rule gives in fact the exact number of positive roots. Indeed, if the number of sign changes is $\sigma$, by applying the Descartes rule to $P(-t)$, we get that the number of negative roots is less than degree$(P) - \sigma$, and thus if all the roots of $P$ are real, Descartes rule gives exactly the number of positive and negative roots.

We thus summarize in Table 1 the different possibilities of sign sequences in the coefficients of $P$. Some sign sequences are impossible: more precisely, we know that $P_0 > 0$, and if $P_2 < 0$ then we can deduce that $K < 0$ and thus that $P_1$ and $P_3$ have opposite signs. Thus only 12 possible sign sequences can occur. The second row of Table 1 gives polynomials having the same sign as $P_i$.

To distinguish between Cases 3a and 3b, which both correspond to two positive roots for $P$, we can compute the difference between the horizontal squared lengths of the segments.

$$D = \tfrac{1}{4}A_1^2A_2^2\big((r_1 - l_1)^2 - (r_2 - l_2)^2\big) = I_1A_2^2 - I_2A_1^2.$$

As noticed above, we are not interested in the whole order on the four endpoints of $\mathcal{A}_1$ and $\mathcal{A}_2$ but in comparing a given endpoint of $\mathcal{A}_1$ with one of $\mathcal{A}_2$. To make this comparison, computing the signs of all the coefficients of $P$ is not always necessary. We use the following observations:

- $P_0$ is positive,
- $P_1$ has the same sign as $J$ (which is simpler),
- when $J$ has been computed, knowing the sign of $P_3$ is equivalent to knowing the sign of $K$,
- if $K$ is positive then $P_2$ is necessarily positive,
- if $J'$ and $J$ have opposite signs, then $P_4$ is positive.

Table 1

| $P_0$ | $P_1$ | $P_2$ | $P_3$ | $P_4$ | Sign changes | Config. |
|---|---|---|---|---|---|---|
| | $J$ | $K+(\ldots)^2$ | $JK$ | $-JJ'+(\ldots)^2$ | | |
| + | + | + | + | + | 0 | 1 |
| + | + | + | + | − | 1 | 2 |
| + | + | + | − | − | 1 | 2 |
| + | + | − | − | − | 1 | 2 |
| + | + | + | − | + | 2 | 3 |
| + | + | − | − | + | 2 | 3 |
| + | − | + | + | + | 2 | 3 |
| + | − | − | + | + | 2 | 3 |
| + | − | + | + | − | 3 | 4 |
| + | − | + | − | − | 3 | 4 |
| + | − | − | + | − | 3 | 4 |
| + | − | + | − | + | 4 | 5 |

Using these facts, Fig. 8 describes an evaluation strategy for comparing $r_1$ and $l_2$, using if possible only the evaluation of small degree expressions. Comparing $l_1$ and $r_2$ is similar. Fig. 9 deals with the comparison of $l_1$ and $l_2$ which is similar to comparing $r_1$ with $r_2$. In both figures, for each new polynomial to evaluate, we give its degree and the number of arithmetic operations (additions, multiplications) needed to evaluate it; of course this number of operations depends on the number of expressions that were previously computed and that are reused. These numbers measure respectively the time and the precision of the computation.

## 8. Arithmetic filters

To implement the predicates described in the previous section, we need to be able to compute the signs of various polynomial expressions exactly.

If we assume, as in Section 2, that the data are fixed sized integers, these computations can be done exactly using some library for exact computation on integers or other suitable number types. However, computing the exact value of an expression with many digits when we are only interested in its sign appears to be a waste in many cases.

A filtering strategy for the evaluation of such predicates [22] has been developed in geometric computing and is detailed in the sequel. Let us assume that the sign of a polynomial expression $Z(\boldsymbol{u})$ must be evaluated (in our case $Z$ is one of our polynomials, $J$ for example), where $\boldsymbol{u}$ denotes its parameters ($\boldsymbol{u} = (\alpha_1, \beta_1, p_1, q_1, s_1, \alpha_2, \beta_2, p_2, q_2, s_2)$ in the case of $J$). The basic idea is to compute an

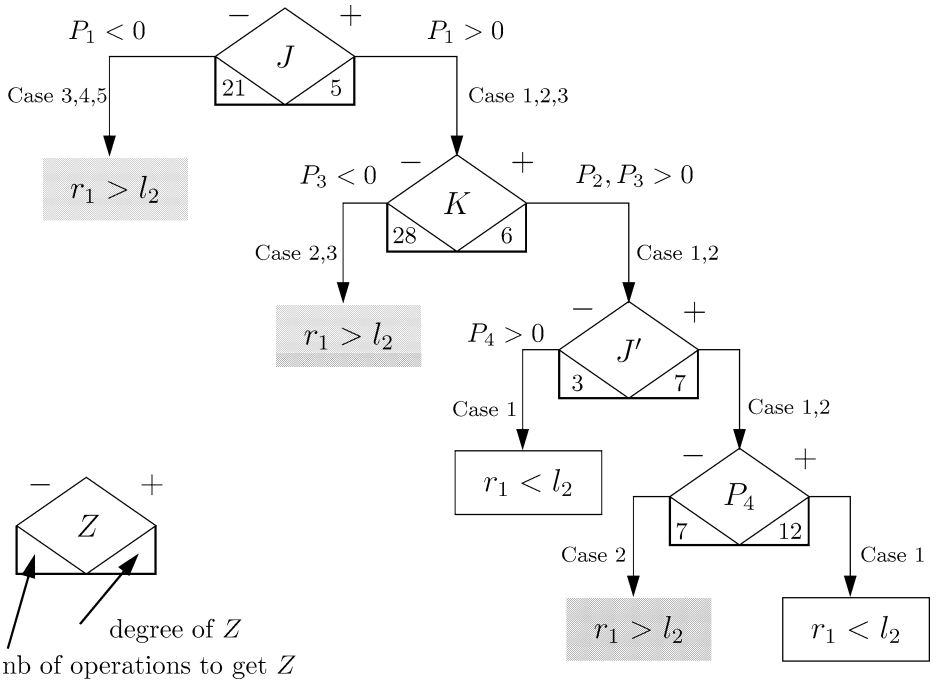



Fig. 8. Strategy of predicate evaluation for comparing $r_1$ and $l_2$.

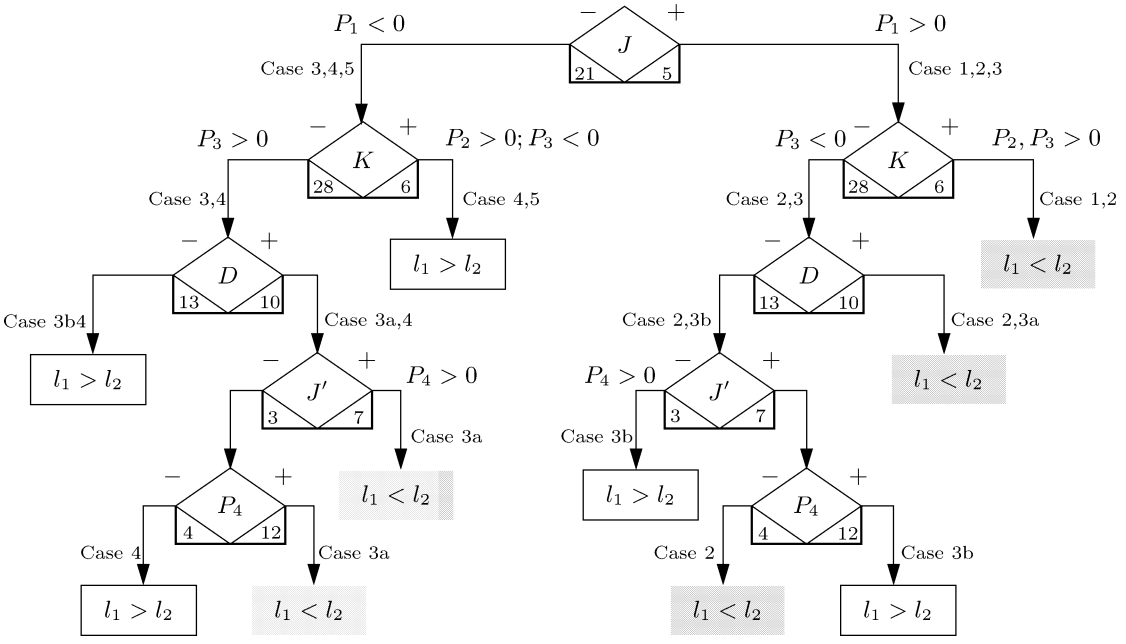



Fig. 9. Strategy of predicate evaluation for comparing $l_1$ and $l_2$.

approximate value $\widehat{Z(\boldsymbol{u})}$ together with a certified error $\varepsilon(Z,\boldsymbol{u})$ on this approximation, which is cheap. If $|\widehat{Z(\boldsymbol{u})}| \geqslant \varepsilon(Z,\boldsymbol{u})$, then the signs of $Z(\boldsymbol{u})$ and $\widehat{Z(\boldsymbol{u})}$ are guaranteed to be the same and the predicate answers safely; otherwise exact computation is performed.

Different kinds of filters can be used, depending on the kind of error computation used.

### 8.1. Static filter

If an upper bound is known on the input $\boldsymbol{u}$ of the predicate $Z$, it is possible to compute a worst case error which does not depend on $\boldsymbol{u}$. This error can be computed in advance and only once for all possible calls of the predicate, we call it the static error and denote it by $\varepsilon(Z)$.

In our implementation, we will assume that our parameters are integers such that $|\alpha_i|, |\beta_i| \leqslant 2^{22}$, $|p_i|, |q_i| \leqslant 2^{24}$, $|\gamma_i| \leqslant 2^{44}$ and $|s_i| \leqslant 2^{47}$. These numbers are stored as `double` according to IEEE 754 standard [12] and rounded computation is used to compute $\widehat{Z(\boldsymbol{u})}$. With such hypotheses we can compute, for any polynomial expression $Z$, an upper bound $M(Z)$ and a maximal error $\varepsilon(Z)$.

More formally, we apply the rules:

- $M(Z \perp Z') = M(Z) \perp M(Z')$, and
- $\varepsilon(Z \perp Z') = \varepsilon(Z)\varepsilon(Z') \perp 2^{-54} \lceil M(Z \perp Z') \rceil$,

where $\perp$ is either the addition or the multiplication and $\lceil V \rceil$ denotes the smallest power of two that is greater than $V$.

We get in this way an upper bound and an error depending only on $Z$: $\forall \boldsymbol{u}$, $|Z(\boldsymbol{u})| \leqslant M(Z)$ and $|Z(\boldsymbol{u}) - \widehat{Z(\boldsymbol{u})}| \leqslant \varepsilon(Z)$.

Table 2 gives the degree $d(Z)$, the upper bound $M(Z)$ and the static error $\varepsilon(Z)$ for all the polynomials $Z$ involved in our predicates. Such a static filter, when it succeeds, introduces no extra cost with respect to the rounded computation since the errors are computed off-line.

### 8.2. Semi-static filter

The static filter fails when $|\widehat{Z(\boldsymbol{u})}|$ is small. This occurs either when the configuration is almost degenerate or when the known upper bound on the input $\boldsymbol{u}$ was over-estimated. Then, another error bound on the approximation $\widehat{Z(\boldsymbol{u})}$ will be computed, not using this bound on $\boldsymbol{u}$. In the case when the configuration is very close to degenerate, this error will still not allow us to conclude on the sign of $Z(\boldsymbol{u})$, and this filter will also fail.

Table 2

| $Z$ | $J$ | $K$ | $J'$ | $D$ | $P_4 = K^2 - 4I_1I_2$ | $P_4 = -4JJ' + J''^2$ |
|---|---|---|---|---|---|---|
| $d(Z)$ | 5 | 6 | 7 | 10 | 12 | 12 |
| $M(Z)$ | $5.32 \cdot 10^{36}$ | $1.12 \cdot 10^{44}$ | $5.62 \cdot 10^{50}$ | $3.54 \cdot 10^{73}$ | $1.65 \cdot 10^{88}$ | $2.48 \cdot 10^{88}$ |
| $\varepsilon(Z)$ | $1.26 \cdot 10^{21}$ | $3.97 \cdot 10^{28}$ | $2.26 \cdot 10^{35}$ | $1.53 \cdot 10^{58}$ | $1.28 \cdot 10^{73}$ | $2.01 \cdot 10^{73}$ |
| $\tau(Z)$ | $4.5 \cdot 10^{-16}$ | $7.8 \cdot 10^{-16}$ | $1.0 \cdot 10^{-15}$ | $6.7 \cdot 10^{-16}$ | $1.5 \cdot 10^{-15}$ | $1.8 \cdot 10^{-15}$ |

One way for computing a semi-static error consists in computing a polynomial $\overline{Z}$ such that applying $\overline{Z}$ to the absolute value $|\boldsymbol{u}|$ of the input yields an upper bound $\overline{Z}(|\boldsymbol{u}|)$ on $Z(\boldsymbol{u})$, tighter than $M(Z)$. This new polynomial can be used to get an error bound $\tau(Z)\overline{Z}(|\boldsymbol{u}|)$ better than $\varepsilon(Z)$, where the relative error $\tau(Z)$ is computed off line and depends only on $Z$. Using the similarity between $Z$ and $\overline{Z}$ often allows a fast computation of $\overline{Z}(\boldsymbol{u})$ and $Z(\boldsymbol{u})$.

More formally we have:

- $\overline{Z+Z'} = \overline{Z} + \overline{Z'}$, $\overline{Z-Z'} = \overline{Z} + \overline{Z'}$, $\overline{Z \cdot Z'} = \overline{Z} \cdot \overline{Z'}$,
- and using IEEE 754 standard: $\tau(Z+Z') = \tau(Z-Z') = \max(\tau(Z), \tau(Z')) + 2^{-53}$ and $\tau(Z \cdot Z') = \tau(Z) + \tau(Z') + 2^{-53}$.

The table gives the values of $\tau$ for the different polynomials involved in our predicates.

### *8.3. Dynamic filter*

The above techniques can be used only for polynomial expressions on integers bounded by some constants known in advance. A simpler technique consists in using some interval arithmetic package to perform the evaluation of $Z(\boldsymbol{u})$. Then the result $\widehat{Z(\boldsymbol{u})}$ comes in the form of an interval which is guaranteed to contain the exact value $Z(\boldsymbol{u})$. Thus, if $0 \notin \widehat{Z(\boldsymbol{u})}$, the predicate can answer safely, otherwise some exact computation must be performed.

These kind of error computation is more expensive than the previous ones but gives also a tighter evaluation of $Z(\boldsymbol{u})$ and thus the number of cases filtered out should be greater. This dynamic filtering can be used in case of failure of the static filter.

## 9. Benchmarks

*Arithmetics*

We tested our strategy and the naive method of Section 3. To this aim we used different kinds of arithmetics and filters:

- `double` means rounded computation of the computer. Times are given as reference, but in difficult situations, the result of the predicate is doubtful.
- `real` stands for the `leda_real` type provided in the LEDA library. It supports the four operations and square root. It contains its own filter which makes some error computation and can evaluate signs exactly.
- GMP or Gnu Multi-precision Package provides exact integer arithmetic.
- "Interval" is the interval arithmetic package available in CGAL library [17,18].
- "Static" means static filtering as described in Section 8 assuming bounds on the input data.
- "Semi-static" means semi-static filtering, described in Section 8 too.

Since the expression of the naive predicate contains division and square root, when no algebraic manipulation is performed on it, it can be evaluated only with "Interval" and "`real`".

The polynomial method can be used with all the arithmetics above. When some filter fails to certify the answer, we switch to another technique.

*Input data*

We have tested several kinds of data generated as follows:

- **rnd22.** We pick at random in $[-M, M]^2$ three points $\Gamma$, $\Gamma'$ and $P$ with $M = 2^{22}$, and construct one of the two arcs defined by the circle of center $\Gamma$ passing through $P$ and by the radical axis of the two circles respectively centered at $\Gamma$ and $\Gamma'$ and passing through $P$. Then, we test if $P$ is the right or left endpoint of the arc and we keep the arc if it matches our needs. The two arcs involved in the predicate are generated independently.
- **rnd16.** Same as above with $M = 2^{16}$. Such an example gives evidence of the efficiency of the semi-static filter in situations where the upper bounds on data are not tight.
- **degenerate.** Same as above (with $M = 2^{22}$), except that the two arcs are not independent, the two points "$P$" have the same abscissa.
- **almost.** Same as the previous one except that the squared radius of the first circle is incremented by one. Thus one of the arcs does not end at $P$ but near $P$.

*Results*

Time performances have been measured with the Unix command `clock`. We have used a PC-Linux with Pentium-III 500 MHz (the compiler is `g++ 2.95.1` with option `-O2 -mcpu=pentiumpro -march=pentiumpro`). Times are given in $\mu$s per predicate evaluation. We also give the percentage of success of the different kinds of filters.

The time for computing with `double` numbers is given as a reference, but of course the results are false in the case of difficult input. We give the percentage of exactness of the predicate with `double`. Since this `double` version of the predicate obviously does not allow to know if a given evaluation is exact or not, we used an exact version to determine this percentage of success.

We give also the percentage of success of the different filtering techniques.

Tables 3–5 show clearly that the polynomial method gives better running times than the naive method even if we use the same arithmetic. Furthermore, the polynomial method allows us to use faster filtering techniques and integer arithmetic for exact computation.

If we compare the Static + Semi-static + Interval + GMP scheme in the polynomial method with the Interval + `real` combination in the naive method, we are 6 times faster in general situations and 15 times faster in degenerate situations. If we compare our strategy to the hazardous `double` evaluation, the time penalty is really small and even an exact evaluation with our strategy is cheaper than an unsafe one using the naive method.

In almost degenerate cases, the dynamic filter using interval arithmetic has a better success rate in the naive method than in the polynomial one. Thus the best strategy is to use static and semi-static filtering to filter out easy cases using the polynomial method, then to use interval arithmetic together with the naive method, and in really difficult cases to use the polynomial method with integer arithmetic.

Table 3
Naive method (running times in $\mu$s)

| Data | | Times | | |
|---|---|---|---|---|
| | | `double` | `real` | Interval + `real` |
| $l_1 \leqslant r_2$? | rnd22 | 0.60 | 23.8 | 2.48 |
| | rnd16 | 0.60 | 23.1 | 2.48 |
| | almost | 0.60 | 136. | 67. |
| | degenerate | 0.60 | 2140. | 2170. |
| $l_1 \leqslant l_2$? | rnd22 | 0.60 | 23.2 | 2.45 |
| | rnd16 | 0.60 | 22.6 | 2.45 |
| | almost | 0.60 | 124. | 38.1 |
| | degenerate | 0.60 | 2145. | 2180. |

Table 4
Polynomial method (running times in $\mu$s)

| Data | | Times | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | `double` | `real` | GMP | Interval + `real` | Interval + GMP | Static + Interval + `real` | Static + Semi-static + Interval + `real` | Static + Semi-static + Interval + GMP | Static + Semi-static + naive Interval + GMP |
| $l_1 \leqslant r_2$? | rnd22 | 0.25 | 20 | 82 | 1.50 | 1.50 | 0.35 | 0.36 | 0.36 | 0.36 |
| | rnd16 | 0.25 | 19 | 78 | 1.50 | 1.50 | 1.40 | 0.41 | 0.41 | 0.41 |
| | almost | 0.25 | 320 | 115 | 229. | 25. | 230. | 227. | 24.3 | 6.80 |
| | degenerate | 0.25 | 1880 | 115 | 1890. | 129. | 1900. | 1900. | 129. | 128. |
| $l_1 \leqslant l_2$? | rnd22 | 0.28 | 27 | 107 | 1.92 | 1.92 | 0.70 | 0.45 | 0.46 | 0.44 |
| | rnd16 | 0.28 | 27 | 102 | 1.92 | 1.92 | 2.11 | 0.57 | 0.56 | 0.56 |
| | almost | 0.28 | 176 | 130 | 104. | 12.4 | 105. | 105. | 13. | 5.64 |
| | degenerate | 0.28 | 2240 | 130 | 2250. | 150. | 2262. | 2250. | 148. | 144. |

Table 5

| Data | | Naive method | | | Polynomial method | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Exactness | Success | | Exactness | Success | | | |
| | | `double` % | Interval % | Exact % | `double` % | Static % | Semi-static % | Interval % | Exact % |
| $l_1 \leqslant r_2$? | rnd22 | 100 | 100 | 100 | 100 | 99 | 100 | 100 | 100 |
| | rnd16 | 100 | 100 | 100 | 100 | 40 | 100 | 100 | 100 |
| | almost | 99 | 96 | 100 | 78 | 0 | 60 | 82 | 100 |
| | degenerate | 8 | 0 | 100 | 17 | 0 | 0 | 0 | 100 |
| $l_1 \leqslant l_2$? | rnd22 | 100 | 100 | 100 | 100 | 84 | 100 | 100 | 100 |
| | rnd16 | 100 | 100 | 100 | 100 | 7 | 100 | 100 | 100 |
| | almost | 99 | 98 | 100 | 99 | 0 | 75 | 93 | 100 |
| | degenerate | 7 | 0 | 100 | 5 | 0 | 0 | 0 | 100 |

## 10. Conclusion

In this paper we studied in detail a geometric predicate needed in the sweep line algorithm for arrangement of circle arcs.

We have shown that techniques from algebraic geometry such as resultant and Bezoutian provide polynomial formula for such predicates. These formulas have been converted in an efficient algorithm for the predicate evaluation which compared favorably to the usual naive evaluation of the predicate. Furthermore, these formulas, as opposed to the naive ones, allow the use of more efficient filters and arithmetics, which results in exactness and efficiency.

In this paper, we only benchmarked isolated predicates. In practice, the predicates would be executed as a part of a whole algorithm. In a concrete implementation, intersection points computed once with the naive method could be stored and reused for several comparisons. However, dealing with robustness issues would then require either computing every intersection point with an exact number type such as `real`, in order to ensure that all comparisons will be performed exactly, or other complex techniques such as filtered constructions [10].

## Acknowledgements

The authors would like to thank Sylvain Pion for helpful discussions on arithmetic filters.

## References

[1] L. Busé, M. Elkadi, B. Mourrain, Generalized resultant over unirational algebraic varieties, J. Symbolic Comput. 29 (2000) 515–526.
[2] J.-D. Boissonnat, F.P. Preparata, Robust plane sweep for intersecting segments, Research Report 3270, INRIA, Sophia Antipolis, France, September 1997.
[3] J.-D. Boissonnat, F.P. Preparata, Robust plane sweep for intersecting segments, SIAM J. Comput. 29 (5) (2000) 1401–1421.
[4] J.-D. Boissonnat, J. Snoeyink, Efficient algorithms for line and curve segment intersection using restricted predicates, in: Proc. 15th Annual ACM Symp. Comput. Geom., 1999, pp. 370–379.
[5] J.-D. Boissonnat, A. Vigneron, Elementary algorithms for reporting intersections of curve segments, in: Proc. 12th Canadian Conference on Computational Geometry, 2000.
[6] The CGAL Reference Manual, 1999. Release 2.0.
[7] J. Dixmier, Quelques aspects de la théorie des invariants, Gazette des mathématiques 43 (1990) 39–64.
[8] M. Elkadi, B. Mourrain, Some applications of Bezoutians in effective algebraic geometry, Rapport de Recherche 3572, INRIA, Sophia Antipolis, France, 1998.
[9] I.Z. Emiris, B. Mourrain, Matrices in elimination theory, J. Symbolic Comput. 28 (1999) 3–44.
[10] S. Funke, K. Mehlhorn, LOOK: A lazy object-oriented kernel for geometric computation, in: Proc. 16th Annual ACM Symp. Comput. Geom., 2000, pp. 156–165.
[11] K.O. Geddes, S.R. Czapor, G. Labahn, Algorithms for Computer Algebra, Kluwer Academic, Norwell, MA, 1992.
[12] IEEE Standard for binary floating point arithmetic, ANSI/IEEE Std 754-1985, New York, 1985. Reprinted in SIGPLAN Notices 22 (2) (1987) 9–25.
[13] J.P.S. Kung, G.-C. Rota, The invariant theory of binary forms, Bulletin (New Series) of the American Mathematical Society 10 (1) (1984) 27–85.
[14] G. Liotta, F.P. Preparata, R. Tamassia, Robust proximity queries: An illustration of degree-driven algorithm design, SIAM J. Comput. 28 (3) (1999) 864–889.
[15] B. Mourrain, N. Stolfi, The Hilbert series of invariants of $Sl_n$, in: G. Jacob, N.E. Oussous, S. Steinberg (Eds.), IMACS SC'93, Lille (France), June 1993, pp. 89–96.
[16] T. Muir, History of Determinants, Dover Reprints, 1960, 5 volumes.
[17] S. Pion, De la géométrie algorithmique au calcul géométrique, Thèse de Doctorat en Sciences, Université de Nice-Sophia Antipolis, France, 1999.
[18] S. Pion, Interval arithmetic: An efficient implementation and an application to computational geometry, in: Workshop on Applications of Interval Analysis to Systems and Control, 1999, pp. 99–110.
[19] B. Sturmfels, Algorithms in Invariants Theory, RISC Series on Symbolic Computation, Springer Verlag, Vienna, 1993.
[20] J.Y. Uspensky, Theory of Equations, McGraw Hill, New York, 1948.
[21] H. Weyl, The Classical Groups, their Invariants and Representations, Princeton University Press, Princeton, NJ, 1939.
[22] C. Yap, Towards exact geometric computation, Computational Geometry Theory and Applications 7 (1) (1997) 3–23.